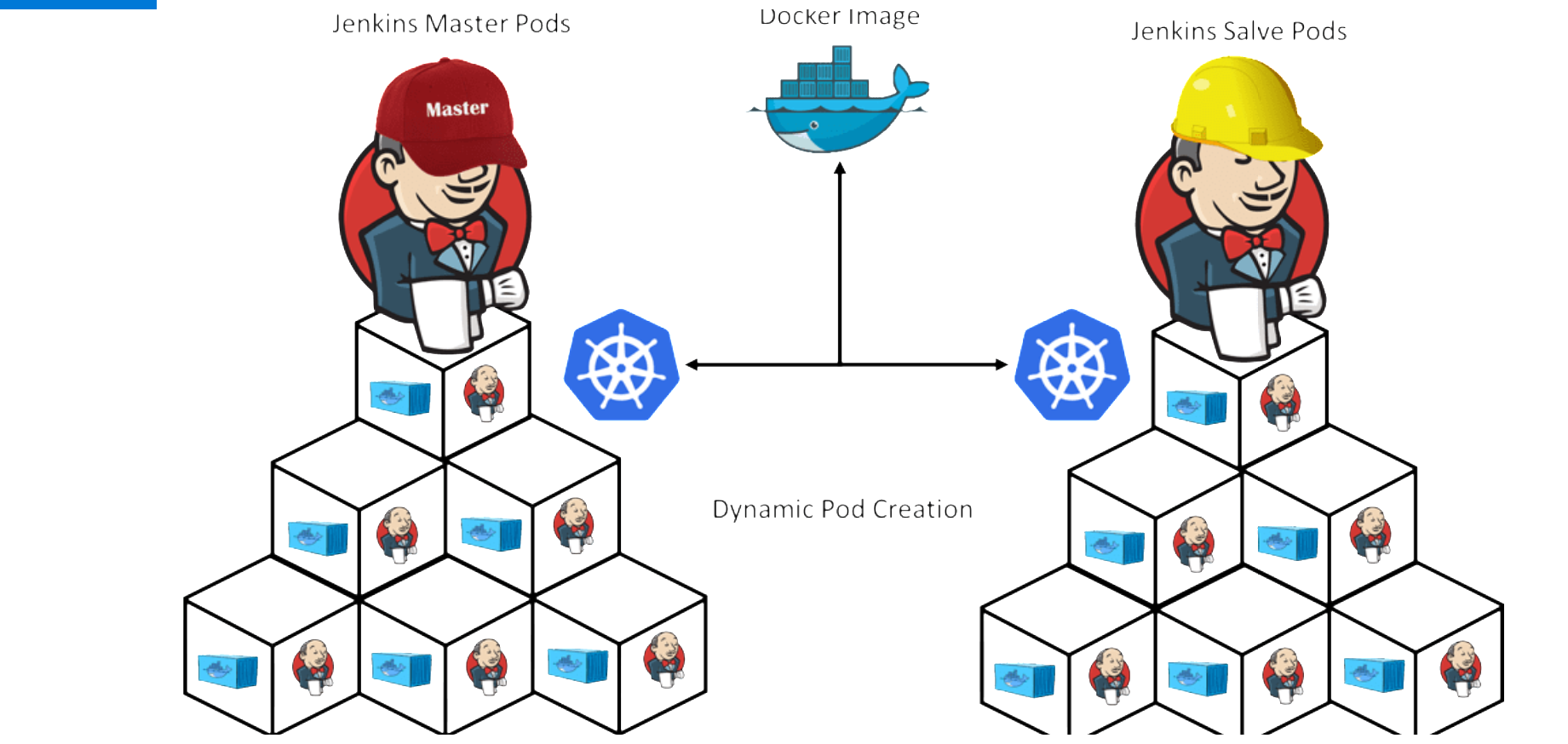
f

[ DevOps- Task-4 ]

* Creation of dynamic distributed jenkins cluster and Performing Task-3 with dynamic jenkins cluster.
* Jenkins is one of the mostly used Continuous Integration tools for build automation due to its capacity in managing a great number of nodes, that are called slaves, with executors for a wide range of tasks related to project build and deploy. Each executor runs Jenkins jobs, alleviating Jenkins server from running all the tasks.
* Jenkins capacity of running jobs should not be constrained by the hardware where it runs, or by the OS, or even by the locality where the jobs need to run. In this way, Jenkins slaves can do these jobs in other machines, in other networks (as long these machines can communicate by HTTP), in other Operating Systems, such as Windows or Solaris, and in other more powerful hardware. For instance, we can compile an application in C++ for Solaris 9 and several other Java 8 applications using Jenkins. Since there is no Java 8 distribution for Solaris 9, setting a slave was crucial to undertake this activity.
* Docker is a paravirtualization solution for Linux to run distributed applications. Due to its success, Docker became a “*de facto”* standard in the software industry, having several images made by the community and stored in its registry. For this reason, several cloud vendors implement Docker solutions in their products, so as Jelastic. In this demo, we'll use some Docker images from Jenkins Swarm and Jenkins Swarm slaves.

1. Create container image that’s has Linux and Other basic configuration required to run Slave for Jenkins. (example here we require kubectl to be configured).

2. When we launch the job it should automatically starts job on slave based on the label provided for dynamic approach.

3. Create a job chain of job1 & job2 using build pipeline plugin in Jenkins

4. Job-1: Pull the GitHub repo automatically when some developers push repo to GitHub and perform the following operations as:

4.1 Create the new image dynamically for the application and copy the application code into that corresponding docker image

4.2 Push that image to the docker hub (Public repository)

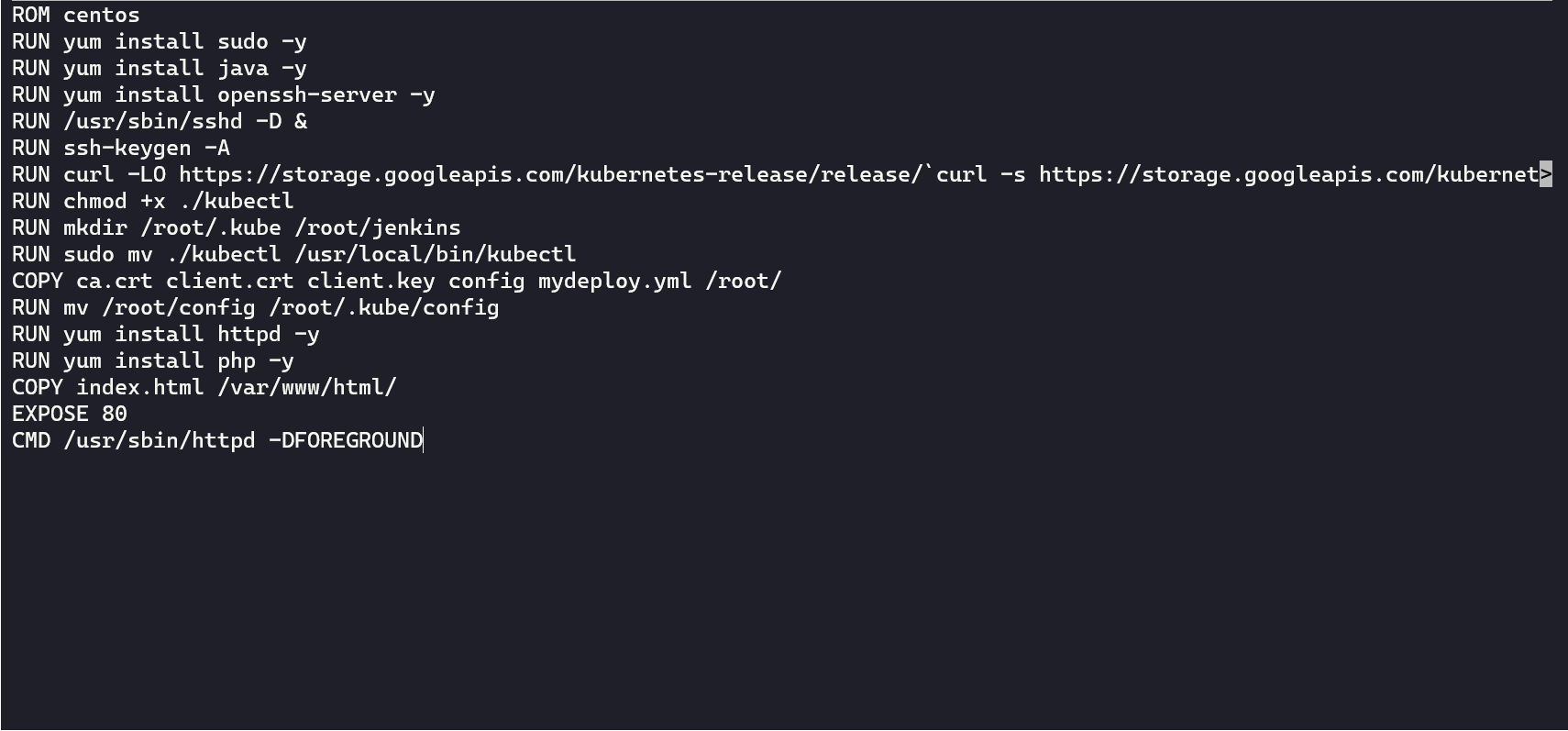
  GitHub code contain the application code and Dockerfile to create a new image.

5. Job2 (Should be run on the dynamic slave of Jenkins configured with Kubernetes kubectl command): Launch the application on the top of Kubernetes cluster performing following operations:

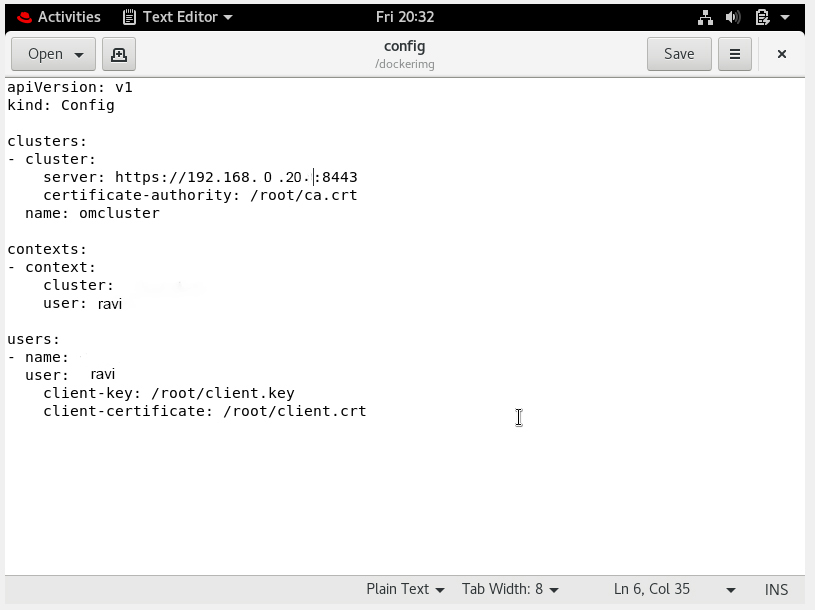
5.1 If launching first time then create a deployment of the pod using the image created in the previous job. Else if deployment already exists then do rollout of the existing pod making zero downtime for the user.

5.2.  If Application created first time, then Expose the application. Else don’t expose it.

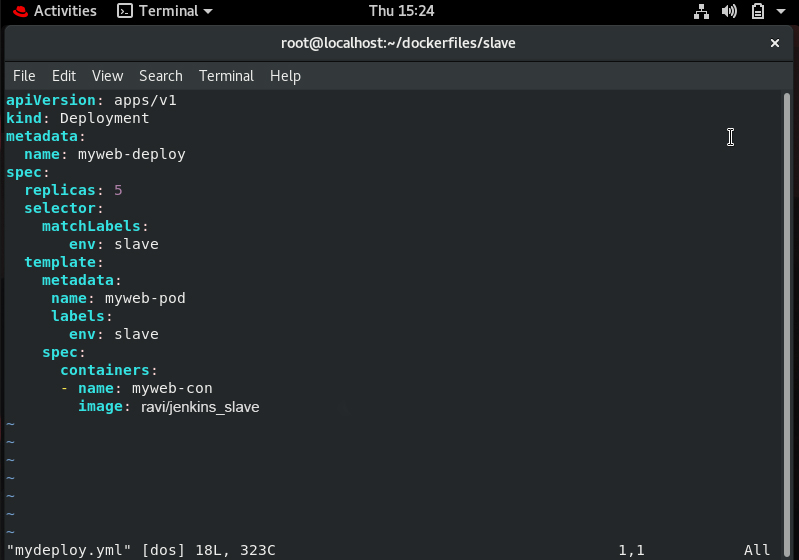
Creating the Container image / Dockerfile :



Now creating Configuration file.

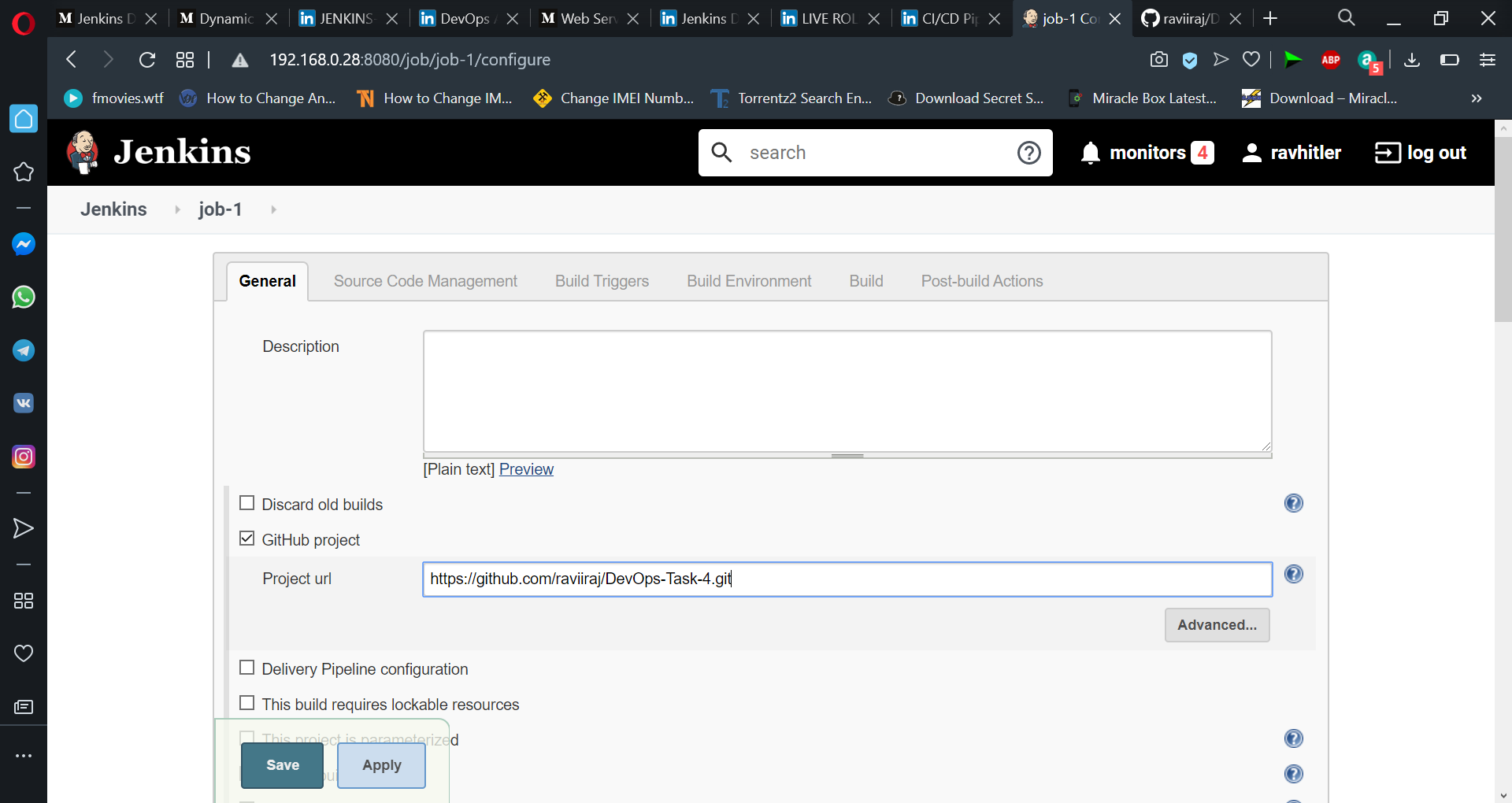


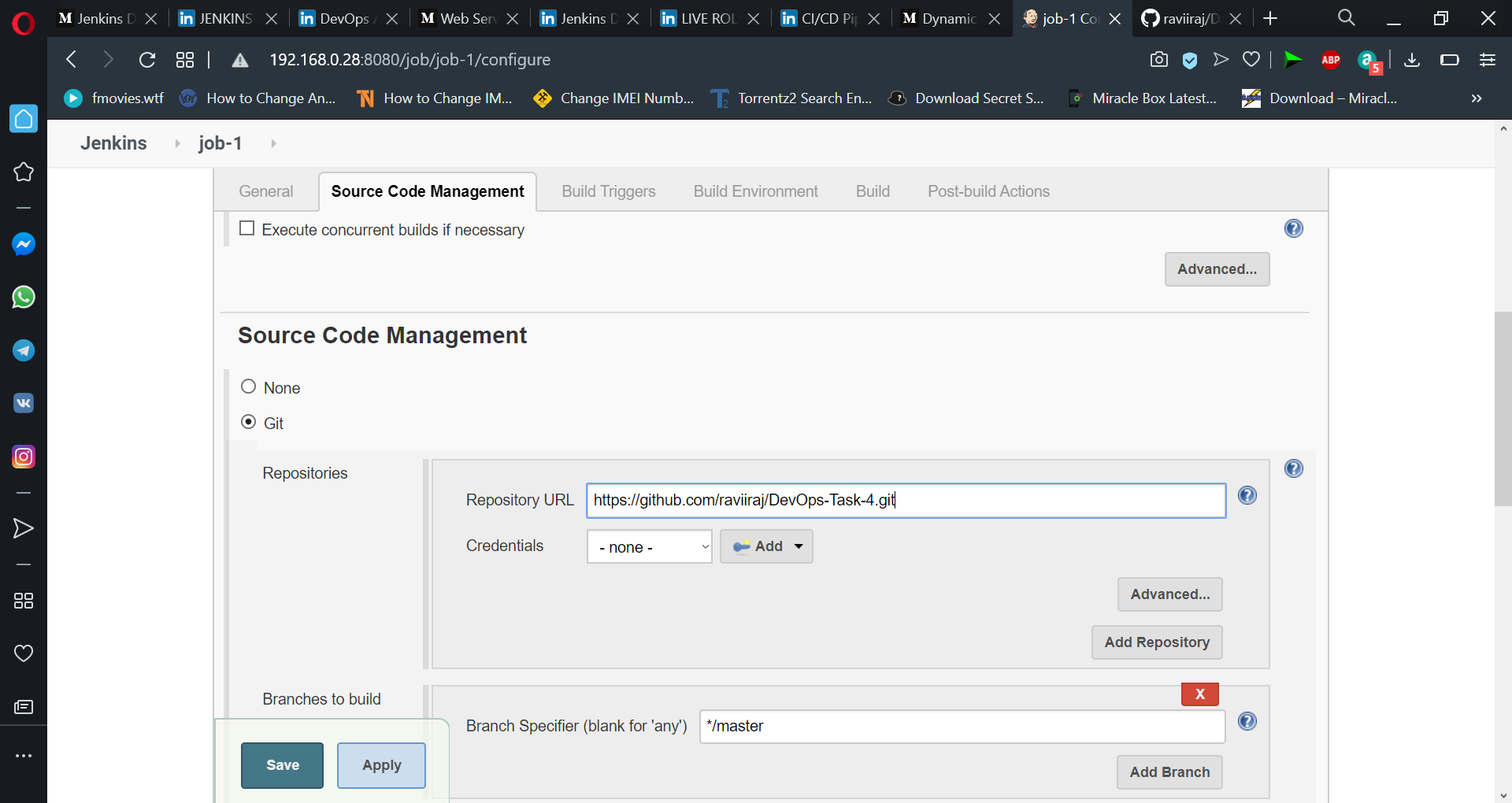
Lastly Deployment File

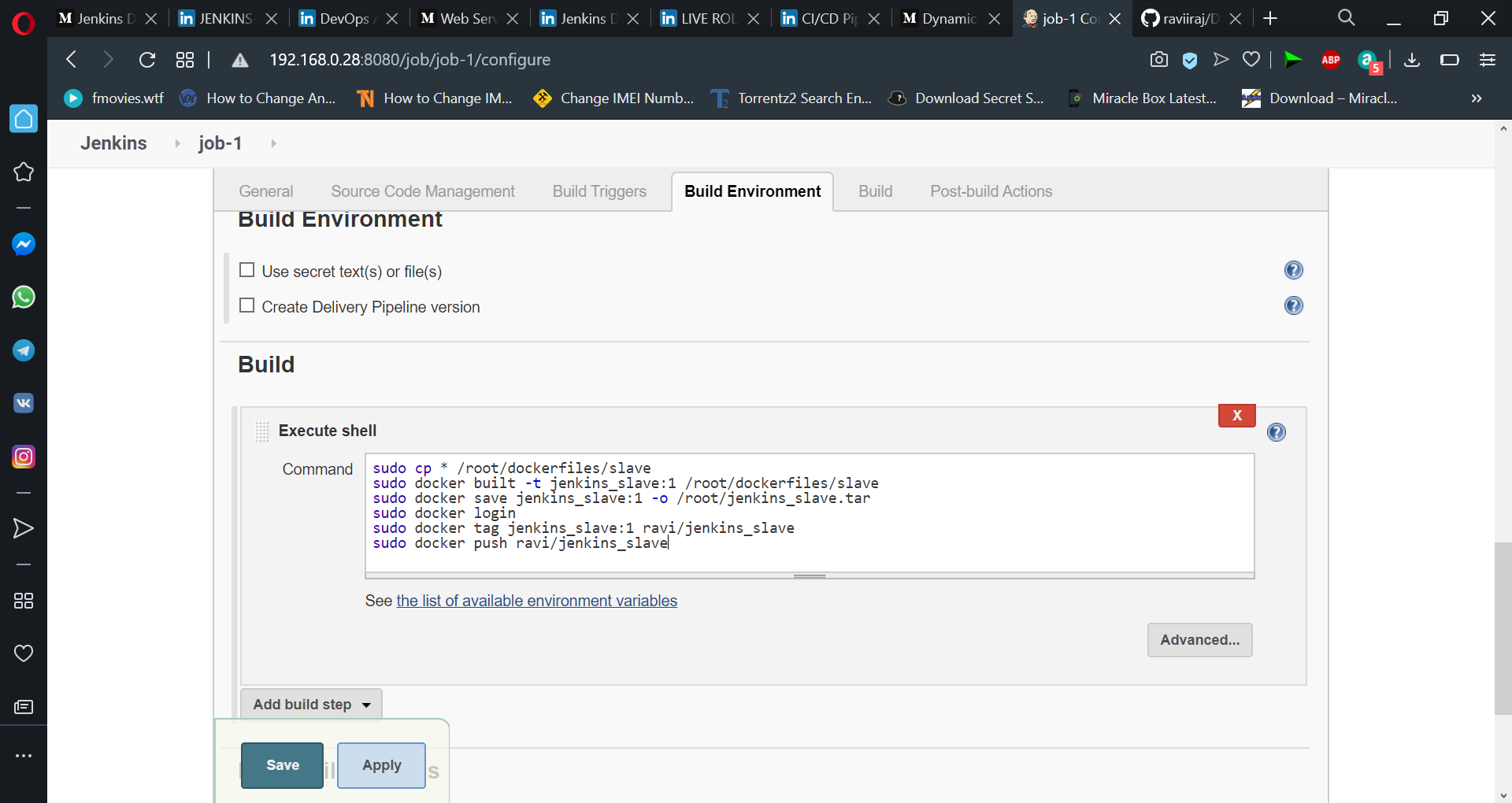


* Create Job-1 in jenkins

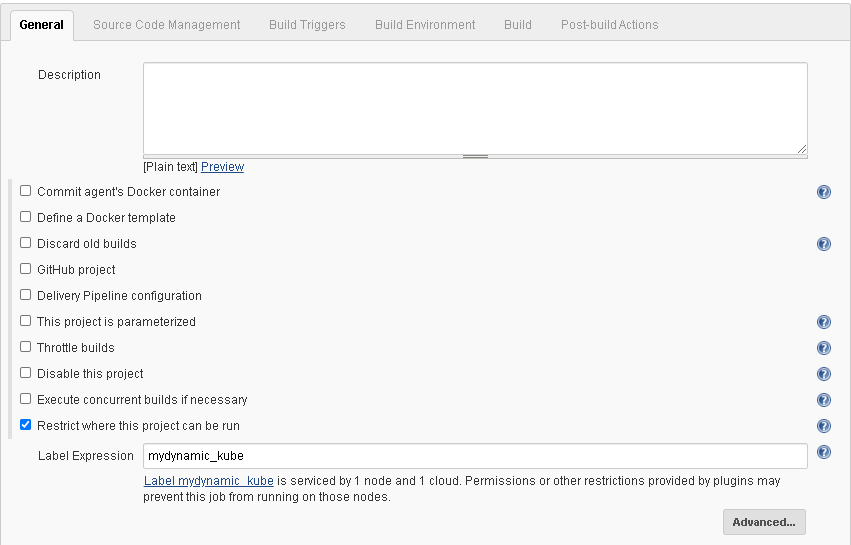
It should pull the GitHub repo automatically when some developers push repo to GitHub.

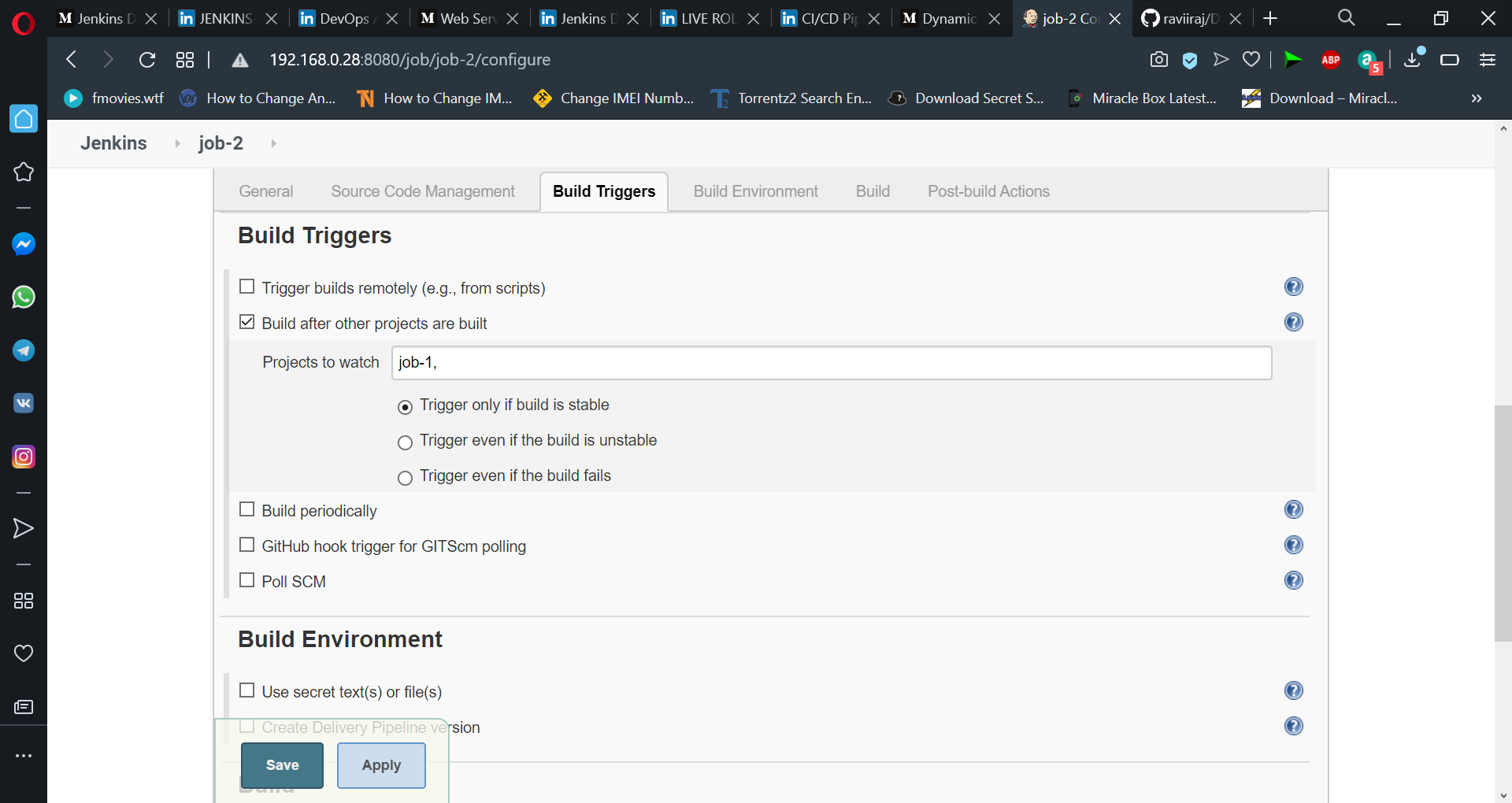


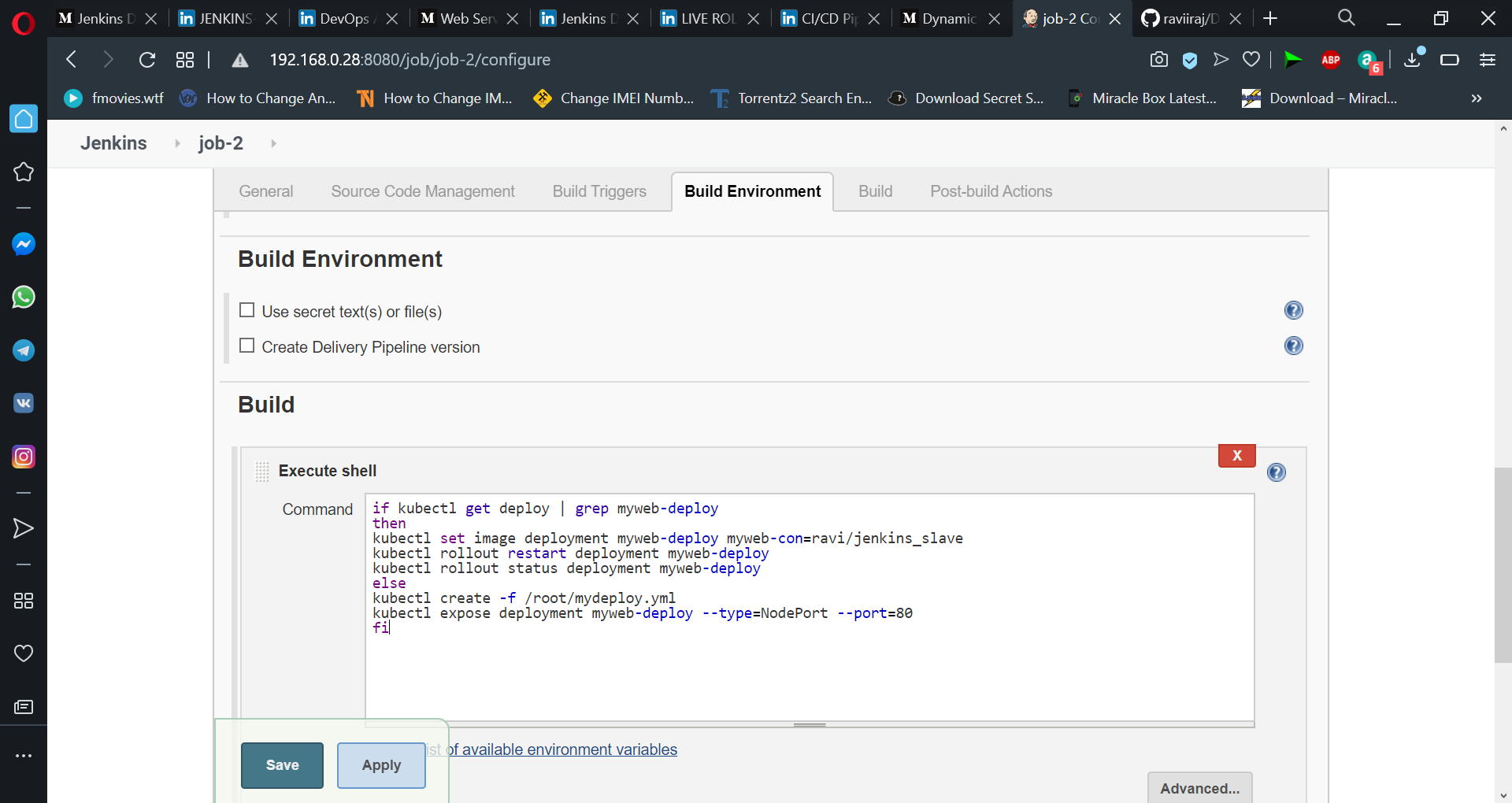




* Job-2 : For this job, We got to server and turn off the docker and run the command export DOCKER\_HOST=ip:port and in slave node edit the docker.service file to -H tcp://0.0.0.0:port.

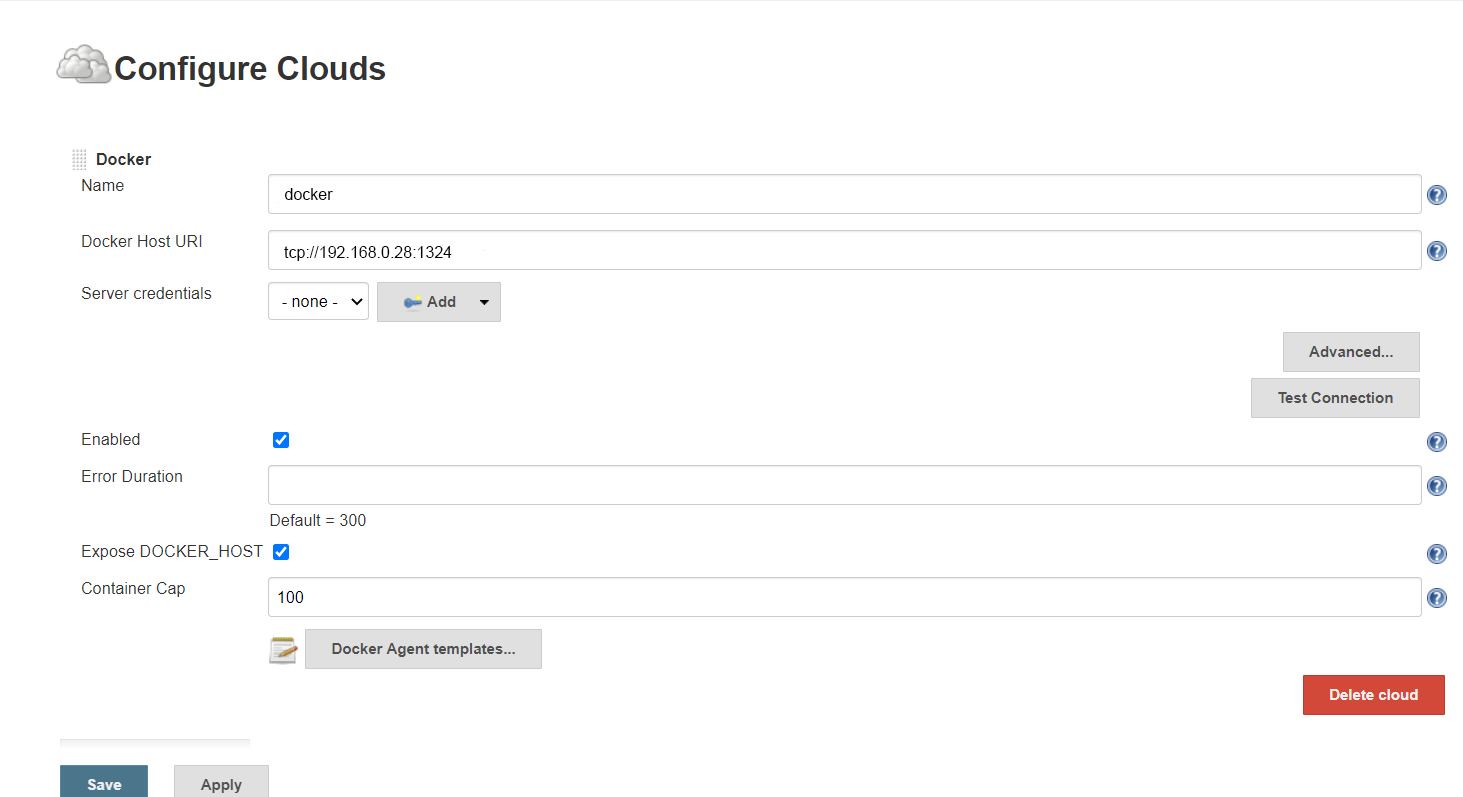


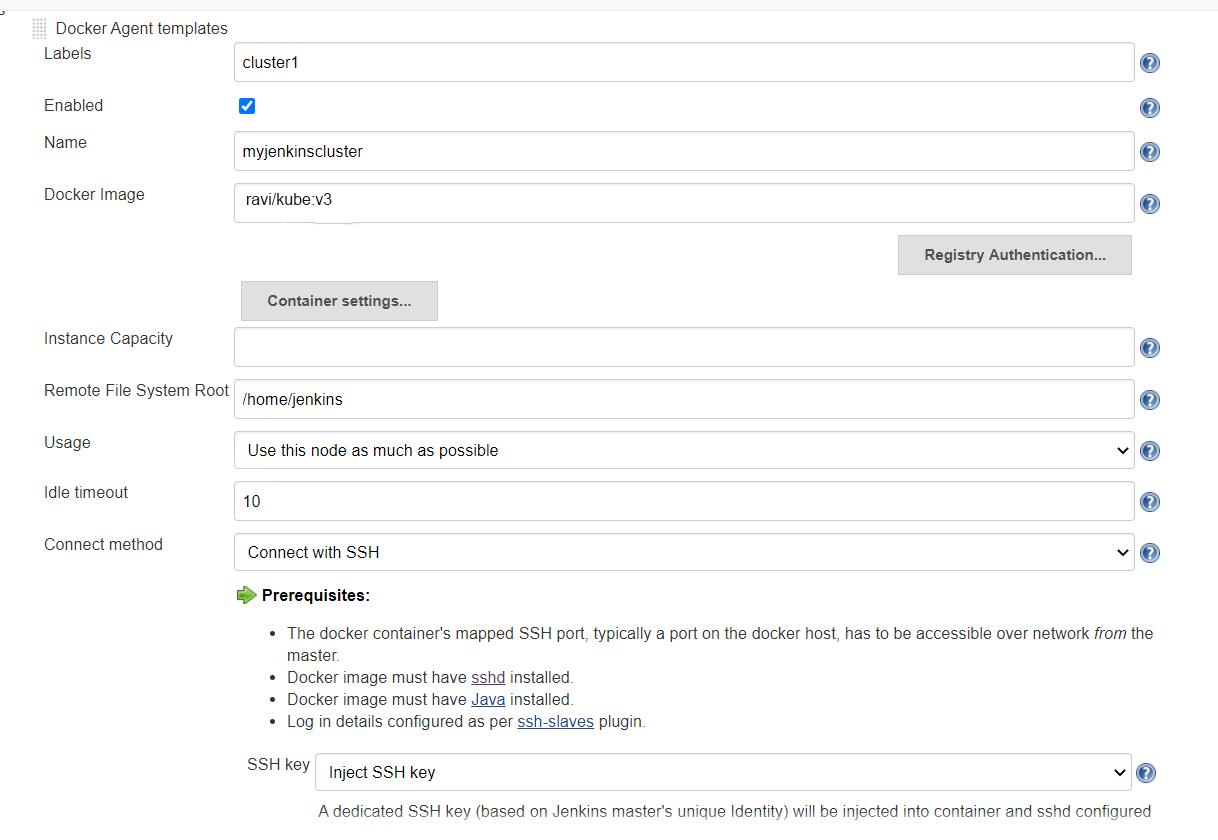




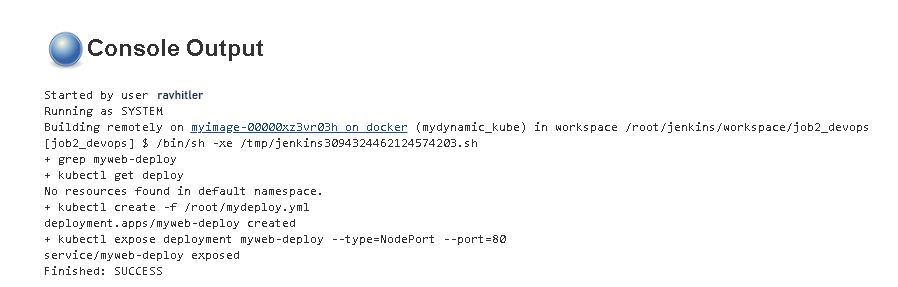
 Configure Cloud for launching Dynamic slave of Jenkins :

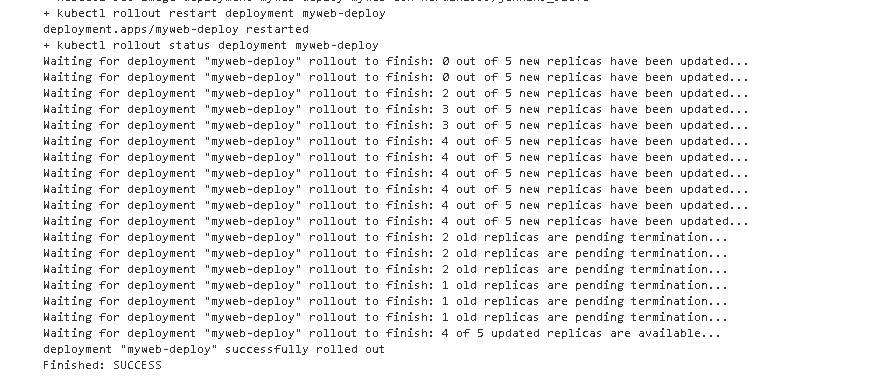
* Plugin required : Docker



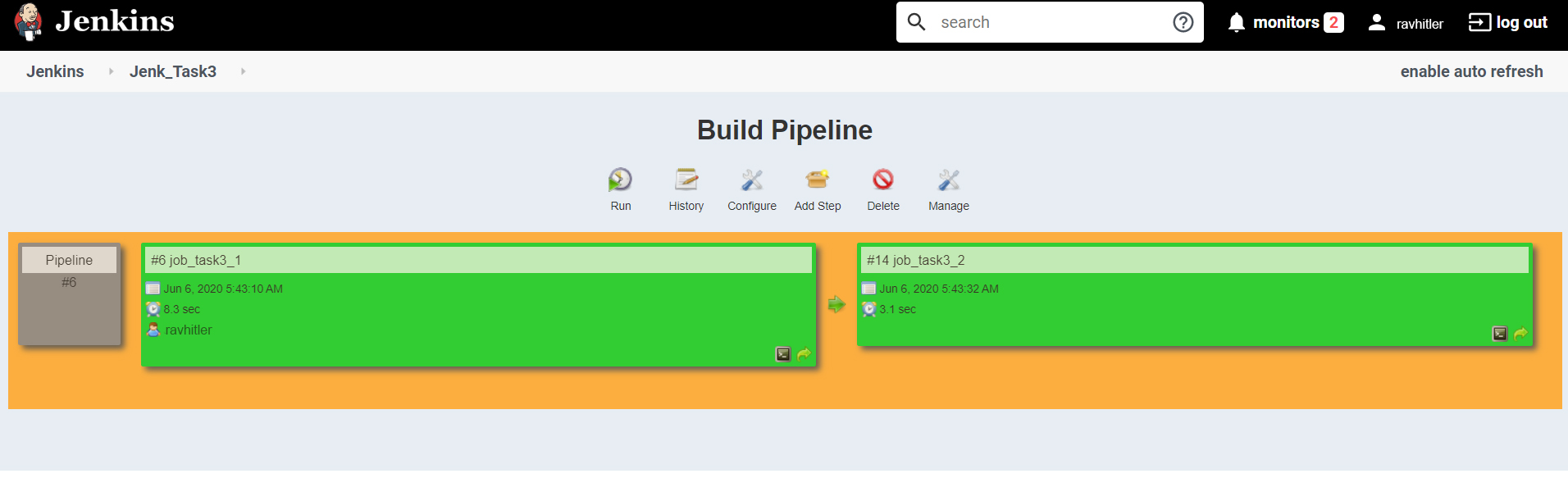


Launch the deployment and expose it for the first time using the image created in the previous job :





**Build Pipeline for job-1 and job-2**

****

**That’s all. Completed task-4 of DevopsAL .**